%% Final Project - Warmup exercise (2-class classifier)

%% Part 1 - Loading the data

load ovariancancer;

%% Part 2 - Feature selection

%% 2.1: Prepare the data

% Set the random number generator to a known state.

% Otherwise, your results may differ.

rng(8000,'twister');

% Partitioning the dataset (for feature selection): 160 points for the

% training set and the remaining 56 for testing.

holdoutCVP = cvpartition(grp,'holdout',56);

dataTrain = obs(holdoutCVP.training,:);

grpTrain = grp(holdoutCVP.training);

%% 2.2: Feature selection (step 1): using a simple filter approach

% Filters are usually used as a pre-processing step in feature selection,

% due to their simplicity and speed.

% A widely-used filter method for bioinformatics data is to apply

% a statistical test separately on each feature, assuming that there is

% no interaction between features.

% For example, we might apply the \_t\_-test on each feature and compare

% \_p\_-value (or the absolute values of \_t\_-statistics) for each

% feature as a measure of how effective it is at separating groups.

dataTrainG1 = dataTrain(grp2idx(grpTrain)==1,:);

dataTrainG2 = dataTrain(grp2idx(grpTrain)==2,:);

[h,p,ci,stat] = ttest2(dataTrainG1,dataTrainG2,'Vartype','unequal');

% In order to get a general idea of how well-separated the two groups are

% by each feature, we plot the empirical cumulative distribution function

% (CDF) of the \_p\_-values:

figure(1), ecdf(p);

xlabel('P value');

ylabel('CDF value')

%% 2.3: Feature selection (step 2): using sequential feature selection

% Use the filter results from the previous section as a

% pre-processing step to select features: sort the features according

% to their p values and select the top 150 features.

[~,featureIdxSortbyP] = sort(p,2);

fs1 = featureIdxSortbyP(1:150);

% Generate a stratified 10-fold partition for the training set:

tenfoldCVP = cvpartition(grpTrain,'kfold',10);

% Apply forward sequential feature selection on these 150 features.

% The function |sequentialfs| provides a simple way (the default option) to

% decide how many features are needed. It stops when the first local

% minimum of the cross-validation MCE (misclassification error) is found.

fun = @(xtrain,ytrain,xtest,ytest) ...

             sum(~strcmp(ytest,classify(xtest,xtrain,ytrain,'quadratic')));

fsLocal = sequentialfs(fun,dataTrain(:,fs1),grpTrain,'cv',tenfoldCVP);

% The selected features are the following:

fs1(fsLocal)

% To evaluate the performance of the selected model with these four features,

% we compute the MCE on the 56 test samples.

testMCELocal = crossval(fun,obs(:,fs1(fsLocal)),grp,'partition',...

    holdoutCVP)/holdoutCVP.TestSize

%% 2.4: Feature selection (step 3): improving sequential feature selection

% The algorithm may have stopped prematurely. Sometimes a smaller MCE is

% achievable by looking for the minimum of the cross-validation MCE over a

% reasonable range of number of features. Let's draw a plot of

% the cross-validation MCE as a function of the number of features for up

% to 50 features.

[fsCVfor50,historyCV] = sequentialfs(fun,dataTrain(:,fs1),grpTrain,...

    'cv',tenfoldCVP,'Nf',50);

figure(2), plot(historyCV.Crit,'o');

xlabel('Number of Features');

ylabel('CV MCE');

title('Forward Sequential Feature Selection with cross-validation');

%% 2.5: Feature selection (step 4): performing actual feature selection

%%%%% ENTER THE VALUE OF N1 HERE!!! %%%%

%

N1 = 27

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fsCVforN1 = fs1(historyCV.In(N1,:));

% Save selected features for later use

save('fsCVforN1.mat', 'fsCVforN1');

% To show these N1 features in the order in which they are selected in the

% sequential forward procedure, we find the row in which they first become

% true in the |historyCV| output:

[orderlist,ignore] = find( [historyCV.In(1,:); diff(historyCV.In(1:N1,:) )]' );

fs1(orderlist);

% To evaluate these N1 features, we compute their MCE for QDA on the test

% set. We get the smallest MCE value so far:

testMCECVforN1 = crossval(fun,obs(:,fsCVforN1),grp,'partition',...

    holdoutCVP)/holdoutCVP.TestSize

%% Part 3 - Starting fresh (with only the selected features)

close all; clear all; clc

load ovariancancer;

load fsCVforN1;

%% 3.1: Partition dataset into 3 groups

% 80% for training and cross validation

% 20% for testing

%%%%% ENTER YOUR CODE HERE!!! %%%%

holdoutCVP = cvpartition(grp,'holdout',43);

dataTrain = obs(holdoutCVP.training,:);

grpTrain = grp(holdoutCVP.training,:);

dataTest = obs(holdoutCVP.test,:);

grpTest = grp(holdoutCVP.test,:);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Part 4 - Building a model (SVM)

X = dataTrain(:,fsCVforN1);

Y = grpTrain;

% Train an SVM classifier using the radial basis kernel.  Let the software

% find a scale value for the kernel function.  It is good

% practice to standardize the predictors.

SVMModel = fitcsvm(X,Y,'Standardize',true,'KernelFunction','RBF',...

    'KernelScale','auto', 'ClassNames', {'Cancer','Normal'});

%% Part 5 - Evaluating the model

%% 5.1: Cross validate the SVM classifier using 10-fold cross validation.

% Perform cross-validation

%%%%% ENTER YOUR CODE HERE!!! %%%%

CVSVMModel1 = crossval(SVMModel,'kfold',10);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Estimate the out-of-sample misclassification rate.

%%%%% ENTER YOUR CODE HERE!!! %%%%

classLoss1 = kfoldLoss(CVSVMModel1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute validation accuracy

%%%%% ENTER YOUR CODE HERE!!! %%%%

validationAccuracy1 = 1 - classLoss1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute validation predictions and scores

[validationPredictions1, validationScores1] = kfoldPredict(CVSVMModel1);

% Display confusion matrix

[Conf\_Mat,order] = confusionmat(grpTrain,validationPredictions1);

disp(Conf\_Mat)

% Compute and display accuracy "per class"

cp1 = classperf(grpTrain,validationPredictions1);

cp1.PositivePredictiveValue

cp1.NegativePredictiveValue

%% 5.2: Cross validate the SVM classifier using holdout (with 20%).

% Perform cross-validation

CVSVMModel2 = crossval(SVMModel,'holdout',0.2);

% Estimate the out-of-sample misclassification rate.

classLoss2 = kfoldLoss(CVSVMModel2)

% Compute validation accuracy

validationAccuracy2 = 1 - classLoss2

% Compute validation predictions and scores

[validationPredictions2, validationScores2] = kfoldPredict(CVSVMModel2);

% Display confusion matrix

[Conf\_Mat,order] = confusionmat(grpTrain,validationPredictions2);

disp(Conf\_Mat)

% Compute and display accuracy "per class"

cp2 = classperf(grpTrain,validationPredictions1);

cp2.PositivePredictiveValue

cp2.NegativePredictiveValue

%% 5.3: Display ROC and compute AUC

% Compute the ROC curve.

SVMModel = fitPosterior(SVMModel);

[~,score\_svm] = resubPredict(SVMModel);

[X2,Y2,T,AUC] = perfcurve(grpTrain,score\_svm(:,1),'Cancer');

% Plot the ROC curve

figure(3), plot(X2,Y2)

xlabel('False positive rate'); ylabel('True positive rate');

title('ROC Curves for SVM, Training dataset')

% Display the area under the curve.

disp(AUC)

%% Part 6 - Testing

%Compute accuracy

testclassLoss = loss(SVMModel, dataTest(:,fsCVforN1), grpTest)

testAccuracy1 = 1 - testclassLoss

% Label the test sample observations.

% Display the results for the observations in the test sample.

[label\_test,score\_test] = predict(SVMModel,dataTest(:,fsCVforN1));

% 3.2: Display confusion matrix

% PASTE YOUR CODE HERE!

[Conf\_Mat\_test,order\_test] = confusionmat(grpTest,label\_test)

disp(Conf\_Mat\_test)

% Compute and display accuracy "per class"

cp3 = classperf(grpTest,label\_test);

cp3.PositivePredictiveValue

cp3.NegativePredictiveValue

% Display ROC and compute AUC

% Compute the ROC curve.

[X\_test,Y\_test,T\_test,AUC\_test] = perfcurve(grpTest,score\_test(:,1),'Cancer');

% Plot the ROC curve

figure(4), plot(X\_test,Y\_test)

xlabel('False positive rate')

ylabel('True positive rate')

title('ROC for Classification by SVM, Test Data Set')

disp(AUC\_test)